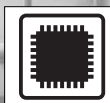
基礎から学ぶ Verilog HDL & FPGA 設計

第４回
順序回路の設計 フリップフロップとカウンタ

中野浩嗣，伊藤靖朗

前回（本誌2007年８月号，pp.129-133）は，算術論理演算回路を設計した．今回は，データを記憶するためのフリップフロップとカウンタを設計する．（筆者）

順序回路は過去の信号の入力に依存して出力が決まる回路です．そのため，順序回路では過去に入力した信号を記憶しておく必要があります．D型フリップフロップ（以下では単にフリップフロップと呼ぶ）をビットのデータを記憶するのに使います．まず，フリップフロップをVerilog HDLで設計しましょう．

● フリップフロップの設計

リスト1はフリップフロップのVerilog HDL記述です．7行目のalways文のイベント・リストは，posedge clkとnegedge resetの二つからなります．ここで，posedgeは立ち上がり（0から1への変化）を，negedgeは立ち下がり（1から0への変化）を意味します．従って，clkが立ち上がった場合，またはresetが立ち下がった場合に次に続く文，この例ではif文が実行されます．このif文では，!resetが1のとき（つまり，resetが0のとき），代入文q <= 0が実行され，さもなければ，q <= dが実行されます．この意味を四つの場合に分けて考えてみましょう．

- posedge clkとnegedge resetが同時に発生した場合．

  イベント発生後のresetは0です．従って，q <= 0が実行され，出力値は0になります．
- posedge clkのみ発生した場合．

  resetが0のとき，出力値は0になります．resetが1ならば，q <= dが実行され，入力をラッチします．
- negedge resetのみ発生した場合．

  resetは0なので，clkの値に関係なく，出力値は0にリセットされます．
- イベントが発生しない場合．

  qに値が代入されないので，qの値は保持されます．

以上の動作を整理すると，**表1**のようになります．クロックclkの立ち上がりに同期して値をラッチするため，「同期ラッチ」と呼ばれます．リセットresetの値が0だとclkに関係なく0を保持するため「非同期リセット」と呼ばれます．また，フリップフロップは**図1**のように表されます．

リスト1　フリップフロップのVerilog HDL記述（ff.v）

```
 1  module ff(clk,reset,d,q);
 2
 3    input clk,reset,d;
 4    output q;
 5    reg q;
 6
 7    always @(posedge clk or negedge reset)
 8      if(!reset) q <= 0;
 9      else q <= d;
10
11  endmodule
```

表1　フリップフロップの動作

入力		出力
clk	reset	q
—	0	0（非同期リセット）
↑	1	d（入力の同期ラッチ）
↑以外	1	直前のq（値の保持）

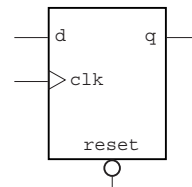



図1　フリップフロップ

KeyWord　フリップフロップ，カウンタ，非同期リセット，同期ラッチ，ノンブロッキング代入文，ブロッキング代入文，組み合わせ回路，完全同期式回路，シミュレーション実行時間，トップ・モジュール

● ブロッキング代入文とノンブロッキング代入文

リスト1の8行目と9行目の代入文では，「<=」を使っていますが，このような代入文を「ノンブロッキング代入文」と呼びます．一連の並んだノンブロッキング代入文は同時に実行されます．例えば，

```
b <= c;
a <= b;
```

は，cの値とbの値がそれぞれbとaに同時に代入されます．

一方，前回（本誌2007年8月号，pp.129-133）行った算術論理演算回路などの組み合わせ回路の設計には，ブロッキング代入文「=」を用います．ブロッキング代入文が並んで記述されている場合，上の行から順番に代入が行われます．例えば，

```
b = c;
a = b;
```

は，cの値がbに代入され，その後，bの値がaに代入されるので，aに代入されるのはcの値となります．従って，上のノンブロッキング代入文の場合とaに代入される値は異なります．一般に，組み合わせ回路の論理を設計する場合はブロッキング代入文が，順序回路の論理を設計する場合はノンブロッキング代入文が推奨されます．

表2　2ビット・カウンタの動作

入力				出力
clk	reset	load	inc	q
—	0	—	—	0（非同期リセット）
↑	1	1	—	d（入力の同期ラッチ）
↑	1	—	1	q+1（インクリメント）
↑	1	0	0	直前のq（値の保持）
↑以外	1	—	—	直前のq（値の保持）

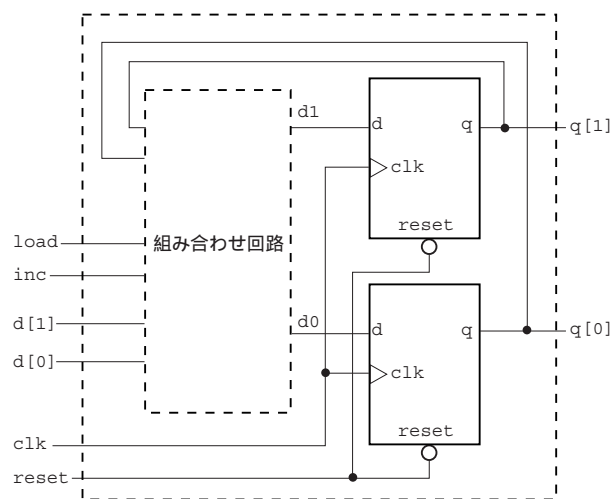



図2　2ビット・カウンタのブロック図

● 2ビット・カウンタの設計

最も簡単な順序回路の一つであるカウンタを設計しましょう．カウンタは2ビットの値を保持するものとし，入力ポートとして，それぞれ1ビットのclk，reset，load，incと，2ビットのdを用います．出力ポートqは，現在のカウンタの値を出力します．resetは，0のときにカウンタの値を0に非同期リセットします．loadが1のとき，clkの立ち上がりでdに入力されているビットの値をラッチします．incが1のとき，clkの立ち上がりでカウンタの値を1増やします（インクリメント）．従って，incが1のとき，出力される値は，00→01→10→11→00→…と変化を繰り返します．loadとincが両方とも0のときは，qの値は変わりません．loadとincが両方とも1になることはないものとします．**表2**は以上の動作をまとめたものです．

フリップフロップと組み合わせ回路を用いて，カウンタを設計できます．記憶する必要があるのは2ビットの値なので，二つのフリップフロップを用いることにします．**図2**はそのブロック図です．カウンタの入力clkとresetは，それぞれ二つのフリップフロップのclkとresetに直結します．また，カウンタの2ビットの出力q（q[0]とq[1]）は，二つのフリップフロップの出力qに接続します．二つのフリップフロップに入力する値d0とd1を，clkの立ち上がりごとにフリップフロップがラッチします．このd0とd1の値を決める組み合わせ回路を設計する必要があります．この組み合わせ回路の入力は，カウンタへの入力load，inc，d（d[0]とd[1]），および，フリップフロップの出力q[0]とq[1]です．これら6ビットの値からd0とd1が決定されます．従って，d0とd1は，load，inc，d[0]，d[1]，q[0]，q[1]の論理式で表せます．以下はその論理式です．

$$d0 = (\overline{load} \wedge \overline{inc} \wedge q[0]) \vee (load \wedge d[0]) \vee (inc \wedge \overline{q[0]})$$

$$d1 = (\overline{load} \wedge \overline{inc} \wedge q[1]) \vee (load \wedge d[1]) \vee (inc \wedge (q[0] \oplus q[1]))$$

この論理式が正しいことをd0について確認してみます．loadとincが両方とも0のとき，d0 = q[0]なので，clkの立ち上がりでq[0]の値は変わりません．loadが1のとき，d0 = d[0]なので，q[0]の値はd[0]となります．incが1のとき，d0 = $\overline{q[0]}$なので，q[0]の値は反

転します．以上より，d0の値が正しく設定され，q[0]の値が意図した通りに変化することが分かります．d1の値が正しく設定されていることも容易に確認できます．**図2**の組み合わせ回路をこの論理式の通りに設計することにより，2ビット・カウンタが完成します．

以上を踏まえて，Verilog HDLで2ビット・カウンタを設計します．**リスト2**はそのVerilog HDL記述です．8行目と9行目でフリップフロップffをインスタンス化します．フリップフロップのポートdに接続されるのは，6行目で宣言されたネットd0とd1です．このd0とd1の値が11行目と12行目のassign文で，前の論理式の通りに決定されています．

図2の2ビット・カウンタは，フリップフロップのclkとresetが外部からの入力clk(グローバル・クロック)とreset(グローバル・リセット)に直結しています．従って，二つのフリップフロップの同期ラッチは同時に起こります．非同期リセットも同時に起こります．このような順序回路を「完全同期式回路」と呼びます．完全同期式回路は，信号の伝搬遅延などの微妙なタイミングを考慮する必要がなく，設計が容易になりバグが発生しなくなります．本連載のCPUは完全同期式回路として設計します．

● 多ビット・カウンタ

上の2ビット・カウンタの設計では，各フリップフロップの入力dの論理を求めました．この方法では，ビット数が増えると組み合わせ回路が複雑になってしまい，設計が困難になります．実は，抽象的な記述によりカウンタを簡単に設計できます．

リスト3はカウンタのVerilog HDL記述です．1行目および4～6行目はモジュール名と入力ポート・出力ポートの宣言です．2行目のparameter文で，Nがデフォルト値16をとるパラメータであることを宣言しています．このモジュールをインスタンス化するときに，Nの値を指定できます．インスタンス化するときに指定しなければ，Nの値はデフォルト値の16となります．指定の方法は後で述べます．

9行目から始まるalways文で，qの値を決定しています．resetが0のときはq <= 0が，loadが1のときはq <= dが，incが1のときはq<=q+1が実行されます．従って，qの値を正しく設定していることは明らかでしょう．このような抽象的かつ平易な記述により，16個のフリップフロップを持つ16ビット・カウンタを設計できます．

● カウンタのシミュレーション

カウンタの動作を確認するためにシミュレーションをしてみましょう．前回と同様に，Sourcesウィンドウ内を右クリックし，「New Source」を選択します．すると，新規ソース・ファイル作成ウィザードが表示されます．ここで，「Verilog Module」を選択し，File Nameを「counter_tb」として，ファイルcounter_tb.vを作成します．そして，**リスト4**のテストベンチのVerilog HDL記述を入力します．

リスト4の4～7行目は，カウンタ・モジュールに接続する入力信号のレジスタ宣言，出力信号のネット宣言，そしてカウンタのモジュール・インスタンス化をしています．

9行目から13行目は，クロックclkを設定します．まず，10行目で初期値を0にして，11行目と12行目のforever文でその後の値の変化を記述します．forever文は同じ文を繰り返し実行するときに用います．この場合は，50単位時間待ち，値を反転するという動作をシミュレーションの最後まで繰り返すという意味となります．1

リスト2　2ビット・カウンタのVerilog HDL記述(counter2.v)

```
 1  module counter2(clk,reset,load,inc,d,q);
 2
 3   input clk,reset,load,inc;
 4   input [1:0]d;
 5   output [1:0]q;
 6   wire d0,d1;
 7
 8   ff ff0(.clk(clk),.reset(reset),.d(d0),.q(q[0]));
 9   ff ff1(.clk(clk),.reset(reset),.d(d1),.q(q[1]));
10
11   assign d0=(~load & ~inc & q[0])|(load & d[0])|(
          inc & ~q[0]));
12   assign d1=(~load & ~inc & q[1])|(load & d[1])|(
          inc & (q[0]^q[1]));
13
14  endmodule
```

リスト3　カウンタのVerilog HDL記述(counter.v)

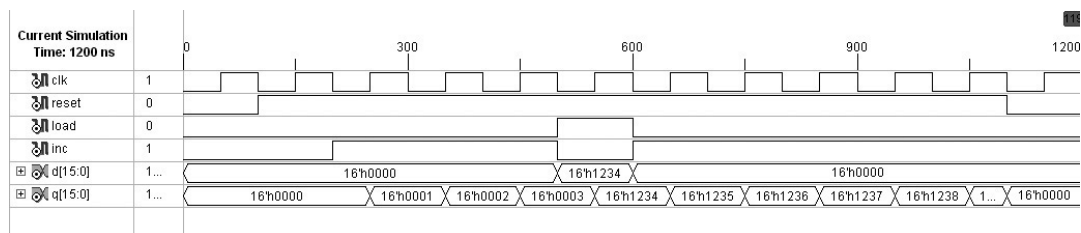
```
 1  module counter(clk,reset,load,inc,d,q);
 2    parameter N=16;
 3
 4    input clk,reset,load,inc;
 5    input [N-1:0]d;
 6    output [N-1:0]q;
 7    reg [N-1:0]q;
 8
 9    always @(posedge clk or negedge reset)
10      if(!reset)q <=0;
11      else if(load)q<=d;
12      else if(inc)q<=q+1;
13
14  endmodule
```

行目で1単位時間を1nsと定義しているので，50nsごとにclkの値が反転します．従って，clkの周波数は1/100nsということになり，開始から50ns，150ns，250ns，350ns，…と，100nsごとにclkの立ち上がりが発生します．16行目以降ではそのほかの信号reset，load，inc，dの値を設定しています．最初はすべての信号の値が0です．100ns後にresetだけ1となり，ほかの信号の値は変化しません．さらに100ns後に，incが1となります．以下同様に，記述された通り信号の値が変化します．

このテストベンチを用いて，カウンタのシミュレーションを行い，動作を確認しましょう．今回は，シミュレーション実行時間を設定します．Processesウィンドウの

リスト4　カウンタのテストベンチのVerilog HDL記述(counter_tb.v)

```
 1  'timescale 1ns / 1ps
 2  module counter_tb;
 3
 4   reg clk,reset,load,inc;
 5   reg [15:0]d;
 6   wire [15:0]q;
 7   counter counter0(.clk(clk),.reset(reset),
         .load(load),.inc(inc),.d(d),.q(q));
 8
 9    initial begin
10      clk=0;
11      forever
12         #50 clk=~clk;
13    end
14
15    initial begin
16      reset=0;load=0;inc=0;d=16'h0000;
17      #100 reset=1;
18      #100 inc=1;
19      #300 inc=0;load=1;d=16'h1234;
20      #100 inc=1;load=0;d=16'h0000;
21      #500 reset=0;
22    end
23
24  endmodule
```

図3 シミュレーション実行時間の設定

「Simulation Behavioral Model」を右クリックし，表示されたメニューの「Properties...」を選択します．すると，「ISE Simulator Properties」のウィンドウが表示されるので，Simulation Run Timeを「1200ns」に設定します(**図3**)．[OK]ボタンをクリックしてこのウィンドウを閉じます．ここで設定した時間まで，または最後に信号の値が変化する時間までシミュレーションが行われます．そして，前回行った通りに，Processesウィンドウの「Simulate Behavioral Model」をダブルクリックすれば，シミュレーションを行えます．**図4**はシミュレーション結果です．550ns後にdの値16'h1234がqにロードされ，それ以降の立ち上がりごとにqの値が1増加していることが分かると思います．

● カウンタのFPGAボードを用いた動作確認

次に，「Spartan-3Eスタータ・キット」のFPGAボードでカウンタの動作を確認してみましょう．ここでは，カウンタをアップカウンタとして使い，押しボタン・スイッチを押すごとにカウントアップし，カウンタの値を8個のLEDで表示する回路を作成します．

連載第1回(本誌2007年4月号，pp.105-114)で全加算器の動作を確認したときは，fa.vに対して直接ピンを割り当てました．今回は，別途モジュールを作成し，カウンタcounter.vをそのモジュール内でインスタンス化し，下位モジュールとする方法を用います．ピン割り当てはこのモジュールに対して行うことになります．

最初にモジュールを作成します．いつもの手順で，counter_top.vをVerilog Moduleとして作成します．このモジュールでは，基本的にモジュールのインスタンス化と配線のみを行います．**リスト5**はこのモジュールのVerilog HDL記述です．入力ポートBTN_NORTH，BTN_EAST，BTN_WEST，BTN_SOUTHは後でピン割り当てを行い，FPGAボードのプッシュ・スイッチに接続します．接続したプッシュ・スイッチを押したときに，入力

図4 カウンタのシミュレーション波形

ポートの値が1になります．そして，値に従って点灯・消灯させるように，出力ポートLEDはFPGAボードのLEDに接続します．5行目はカウンタのモジュール・インスタンス化をしています．

リスト3では，16ビット・カウンタでしたが，ボードのLEDは8個しかないので，8ビット・カウンタにします．そこで，モジュール名とインスタンス名の間に#(数値)と記述することで，パラメータの値を指定します．この場合，モジュールcounterのパラメータNの値が8となります．パラメータが複数ある場合は#(数値1，数値2，…)と記述することでその値を指定できます．入出力ポートに関して，クロックclkはBTN_EASTに接続します．また，リセットresetは0のときにリセットが実行されるように設計しました．BTN_SOUTHに接続したプッシュ・スイッチを押したときにリセットが実行されるよう論理を反転して接続しています．また，BTN_NORTHをload，BTN_WESTをincに接続し，値8'h55をdに入力します．

loadが1のときclkの立ち上がりが起こるとdの値がカウンタにロードされます．BTN_NORTHに接続したプッシュ・スイッチを押した状態で，BTN_EASTに接続したプッシュ・スイッチを押すと，カウンタの値が8'h55にセットされます．またincが1のとき，clkの立ち上がりでカウンタをインクリメントします．従って，BTN_WESTに接続したプッシュ・スイッチを押した状態で，BTN_EASTに接続したプッシュ・スイッチを押すとカウンタ値がインクリメントされます．カウンタの出力qは出力ポートLEDに接続します．

モジュールcounter_top.vを作成後，このモジュールをトップ・モジュールに設定します．トップ・モジュールは，回路の階層構造の根となるモジュールです．トップ・モジュールの入出力ポートがFPGAのピンに接続されます．トップ・モジュールとして設定するため，Sourcesウィンドウの上部のリストから「Synthesis/Implementation」を選択し，counter_top(counter_top.v)を右クリックし「Set as Top Module」を選択します(**図5**)．

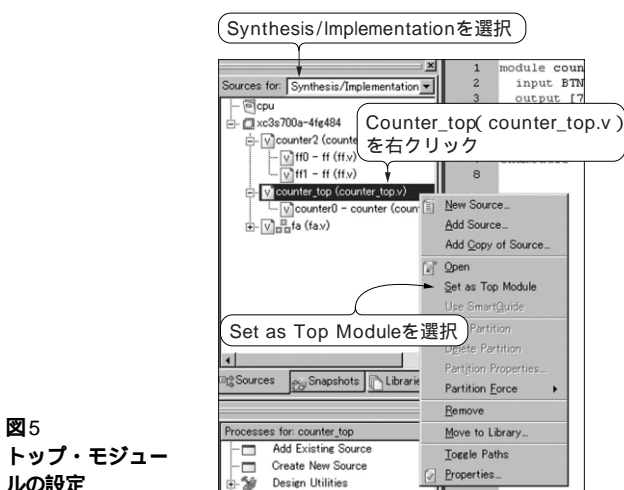
次に，ピン割り当ての設定をするためにユーザ制約ファイルを作成し，モジュールに関連付けをします．連載第1回で説明した方法で，Spartan-3Eスタータ・キット用のユーザ制約ファイルcounter_top.ucfを作成し，**リスト6**を入力します．このときISE WebPACKでは，1プロジェクトの中にユーザ制約ファイルは一つしか追加できないことに注意します．従って，別のユーザ制約ファイルがすでにプロジェクトにある場合は，今あるユーザ制約ファイルをプロジェクトから削除してから新しく追加する必要があります．

図5 トップ・モジュールの設定

リスト5 トップ・モジュールのVerilog HDL記述
(counter_top.v)

```
1  module counter_top(BTN_NORTH,BTN_EAST,
                   BTN_WEST,BTN_SOUTH,LED);
2   input BTN_NORTH,BTN_EAST,BTN_WEST,BTN_SOUTH;
3   output [7:0]LED;
4
5   counter#(8)counter0(.clk(BTN_EAST),
        .reset(~BTN_SOUTH),.load(BTN_NORTH),
        .inc(BTN_WEST),.d(8'h55),.q(LED));
6  endmodule
```

リスト6 トップ・モジュールのユーザ制約ファイル
(counter_top.ucf；Spartan-3Eスタータキット用)

```
 1  #PUSH SWITCH
 2  NET"BTN_NORTH" LOC = "V4"|IOSTANDARD =
      LVTTL|PULLDOWN;
 3  NET"BTN_EAST" LOC = "H13"|IOSTANDARD =
      LVTTL|PULLDOWN;
 4  NET"BTN_WEST" LOC = "D18"|IOSTANDARD =
      LVTTL|PULLDOWN;
 5  NET"BTN_SOUTH" LOC = "K17"|IOSTANDARD =
      LVTTL|PULLDOWN;
 6
 7  #LED
 8  NET"LED<7>" LOC = "F9"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
 9  NET"LED<6>" LOC = "E9"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
10  NET"LED<5>" LOC = "D11"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
11  NET"LED<4>" LOC = "C11"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
12  NET"LED<3>" LOC = "F11"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
13  NET"LED<2>" LOC = "E11"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
14  NET"LED<1>" LOC = "E12"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
15  NET"LED<0>" LOC = "F12"|IOSTANDARD =
      LVTTL|SLEW = SLOW|DRIVE = 8;
```

リスト7　トップ・モジュールのユーザ制約ファイル
(counter_top.ucf；Spartan-3A スタータキット用)

```
 1  #PUSH SWITCH
 2  NET"BTN_NORTH" LOC = "T4"|IOSTANDARD =
        LVTTL|PULLDOWN;
 3  NET"BTN_EAST" LOC = "T16"|IOSTANDARD =
        LVTTL|PULLDOWN;
 4  NET"BTN_WEST" LOC = "U15"|IOSTANDARD =
        LVTTL|PULLDOWN;
 5  NET"BTN_SOUTH" LOC = "T15"|IOSTANDARD =
        LVTTL|PULLDOWN;
 6
 7  #LED
 8  NET"LED<7>"LOC = "W21"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
 9  NET"LED<6>"LOC = "Y22"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
10  NET"LED<5>"LOC = "V20"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
11  NET"LED<4>"LOC = "V19"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
12  NET"LED<3>"LOC = "U19"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
13  NET"LED<2>"LOC = "U20"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
14  NET"LED<1>"LOC = "T19"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
15  NET"LED<0>"LOC = "R20"|IOSTANDARD =
        LVTTL|SLEW = QUIETIO|DRIVE = 4;
```

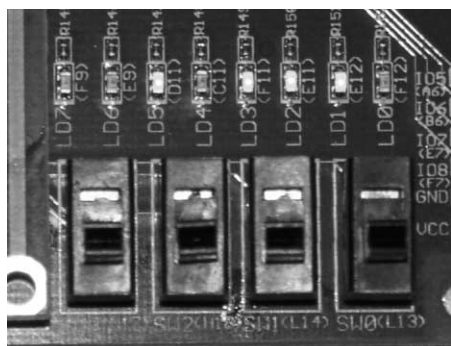



図6　カウンタの値を点灯するLED
2進8桁のカウンタの値をLEDの点灯で表している．点灯が1で消灯が0．

また，**リスト7**はSpartan-3A スタータ・キット用のユーザ制約ファイルです．Spartan-3A スタータ・キットは，Spartan-3E スタータ・キットより大規模のFPGA「XC3S700A-FG484」を搭載しています．ボードの構成の変更はほとんどなく，ピン割り当てだけが異なります．ピン割り当ての詳細はSpartan-3E スタータキット・ボードユーザーガイド(1)，または，Spartan-3A スタータキット・ボードユーザーガイド(2)を参照ください．

連載第1回で説明した手順でビット・ファイル counter_top.bit を作成し，このビット・ファイルをFPGAにダウンロードします．ダウンロードが完了したら，現在のカウンタの値が2進数でLEDに表示されます(**図6**)．**図7**の左側のプッシュ・スイッチを押した状態(つまり，

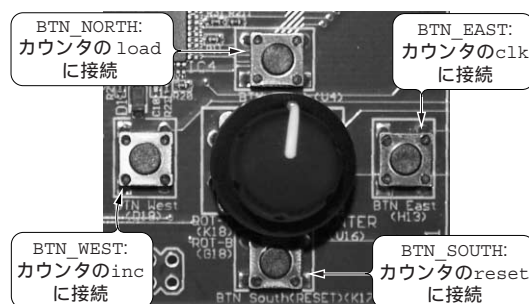



図7　プッシュ・スイッチ
左側のスイッチを押しながら右側のスイッチを押すとカウントアップし，上側のスイッチを押しながら右側のスイッチを押すとカウント値に8'h55がセットされる．また，下側のスイッチを押すとカウント値が0にリセットされる．

inc=1)で右側のプッシュ・スイッチを押す(つまり，clkの立ち上がり)とカウンタの値が増えます．上側のプッシュ・スイッチを押した状態(つまり，load=1)で右側のプッシュ・スイッチを押すと，カウンタの値が8'h55，つまり，2進数で8'b01010101になります．また，下側のスイッチを押すと，resetが0になるので，カウンタの値が0にリセットされ，全LEDが消灯します．一度スイッチを押しただけなのに，複数回カウントしたようにLEDの点灯が変化することがあります．これは，スイッチが切り替わるごく短い時間に，何度もON/OFFが繰り返されるチャタリングと呼ばれる動作が原因です．このチャタリングを防ぐ方法は，次回以降に説明します．

次回は，ステート・マシンを設計します．

参考・引用*文献

(1) Spartan-3E スタータキットボードユーザーガイド.
http://japan.xilinx.com/bvdocs/userguides/ug230.pdf
(2) Spartan-3A スタータキットボードユーザーガイド.
http://japan.xilinx.com/bvdocs/userguides/ug330.pdf

なかの・こうじ
いとう・やすあき
広島大学大学院工学研究科

<著者プロフィール>
中野浩嗣．1992年大阪大学大学院博士後期課程修了．工学博士．一つの民間企業，二つの大学を経て，2003年より，広島大学教授．
伊藤靖朗．2003年北陸先端科学技術大学院大学博士前期課程修了．現在，広島大学助教．